DOCUMENT RESUME

ED 232 856 SE 042 630

AUTHOR Novak, Gordon S., Jr.
TITLE Cognitive Processes and Knowledge Structures Used in Solving Physics Problems. Final Technical Report.
INSTITUTION Texas Univ., Austin. Dept. of Computer Science.
SPONS AGENCY National Science Foundation, Washington, D.C.
REPORT NO NSF/SED-82032
PUB DATE 82
GRANT SED-7912803
NOTE 31p.; For related documents, see SE 042 631-635.
PUB TYPE Reports - Research, Technical (143)

EDRS PRICE MF01/PC02 Plus Postage.
DESCRIPTORS *Cognitive Processes; *Cognitive Structures; *College Science; *Computer Programs; Higher Education; High Schools; *Physics; *Problem Solving; Science Education; *Secondary School Science
IDENTIFIERS National Science Foundation; Science Education Research

ABSTRACT

Cognitive processes and knowledge structures used in physics problem-solving at the high school or freshmen college level were investigated by analysis of expert and novice human problem-solving behavior and by development of computer programs which can solve informally-stated physics problems. Computer representations for the information developed by a problem solver while solving a problem, a computer language for specifying such representations (GLISP), and a compiler for translating programs written in terms of such representations into executable form were developed. In addition, a set of rules for problem analysis were also developed. These rules determine the general context of a problem, fill in features missing from the problem statement, and connect the features of the problem with the appropriate object models and methods for solving the problem. These representations and rules have been used successfully to analyze mechanics problems. Current research involves working on methods by which a program, starting with general formulations of physical laws, can learn expertise in solving particular kinds of problems through practice. (An example of knowledge representation for units of measurement and an example of problems and problem analysis rules are appended to this report). (Author/JN)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

NSF/SED-82032

Cognitive Processes and Knowledge Structures Used in Solving Physics Problems

ED232856

U.S. DEPARTMENT OF EDUCATION
NATIONAL INSTITUTE OF EDUCATION
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
This document has been reproduced as received from the person or organization originating it.
Minor changes have been made to improve reproduction quality.
Points of view or opinions stated in this document do not necessarily represent official NIE position or policy.

Final Technical Report
NSF Grant SED-7912803
9/79 - 9/82

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
National Science Foundation
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

Gordon S. Novak Jr.
Computer Science Department
University of Texas at Austin
Austin, Texas 78712

1. Summary

We have investigated the cognitive processes and knowledge structures used in physics problem solving both by analysis of expert and novice human problem solving behavior and by development of computer programs which can solve informally stated physics problems. We have developed computer representations for the information which is developed by a problem solver while solving a problem, a computer language for specifying such representations, and a compiler for translating programs written in terms of such representations into executable form. We have developed a set of rules for problem analysis; these rules determine the general context of a problem, fill in features missing from the problem statement, and connect the features of the problem with the appropriate object models and methods for solving the problem. These representations and rules have been used successfully to analyze mechanics problems. Finally, we are working on methods by which a program, starting with general formulations of physical laws, can learn expertise in solving particular kinds of problems through practice.

2. Introduction

The goal of our research has been to understand the cognitive processes and knowledge structures which are used in solving physics problems at the high school or college freshman level. Understanding of these cognitive processes and knowledge structures can be beneficial in several ways. First, it can enable computer programs to be written which can solve problems in physics and related technical domains automatically,

SE042630

without requiring that users of such programs be expert in physics. Second, it can enable more effective teaching of physics, since the component processes of physics problem solving are made explicit. Finally, it can provide the basis for intelligent computer-aided instruction (CAI) programs which understand the problem solving process, can answer questions about solutions of example problems, and can analyze failures in student problem solving.

Our research indicates that the common perception that physics is a difficult subject is justified: there is a great deal of knowledge involved in solving physics problems. Moreover, this knowledge is often taught implicitly, by example, rather than explicitly. Explication of the knowledge involved in problem solving should help both teachers and students.

Our research methodology is to codify knowledge about physics problem solving in the form of working computer programs. This provides a solid test of the completeness and soundness of the problem solving methods developed. In a sense, the computer serves as the dumbest possible student: if a procedure has been explained so well that the computer can follow it, we can be assured that no steps have been left out and that the theory contains no "homunculus" in which significant problem-solving expertise is buried.

In the following sections, we discuss our work on knowledge representation, problem analysis, and machine learning of problem solving expertise. These discussions are somewhat abbreviated; more detail is contained in the copies of papers, knowledge bases, and computer programs which are appended to this report.

3. Knowledge Representation

Knowledge representation is the means by which various kinds of knowledge are represented within a computer program. Of course, the details of knowledge representation in a computer program will be entirely different from the details of representation in a human mind. However, at an appropriate higher level of description there will be much similarity. We have freely used introspection to provide insight into ways of representing knowledge within programs; we believe that skill in the selection and use of specialized representations is a significant component of problem solving expertise, and that computer programs must possess similar skills if they are to attain similar levels of capability.

Many different kinds of knowledge must be represented in solving a physics problem. These include not only representations of objects, variables, and equations, but also representations of the constructs introduced by the problem solver. The latter include models of the general physical environment, models of objects as canonical "physics objects" (e.g., "point mass"), models of situations (e.g., "uniform acceleration from rest") and models of constructs used in setting up the problem (e.g., coordinate systems). Diagrams which are drawn by people when solving problems indicate that they must be using representations with similar content.

In addition to these representations involving a particular problem, there must be representations of problem-solving knowledge; some of this knowledge will be in the form of rules, and some will be in the form of procedures.

In order to represent all of these different kinds of knowledge, we found it necessary to develop a language for describing representations and for translating programs written in terms of the described objects into executable form. This detour was undertaken with some reluctance, since it did take time; however, we were unable to find an existing package which had the necessary power and efficiency. The language we developed, GLISP, meets these needs and has served well as an implementation language for components of the problem solving system. In addition, GLISP has been made available to a number of other researchers.

For representing problem analysis rules, we adopted the rule language provided by MRS [1]; this rule language is based on the first order predicate calculus, but also has the ability to be interfaced to our other representations. We made some additions to MRS for our particular needs. The representations we use are described in more detail in the papers appended to this report; a set of problem analysis rules written in MRS form is also included.

4. Problem Analysis

Our analysis of textbook expositions, textbook problems, and the problem-solving protocols of experts and novices has clearly shown that problem solving does not consist of applying all the applicable physical laws in the context of a particular problem. Instead, the primary activity in solving a physics problem is the construction of an *analysis model* for the problem. This model will view the actual objects of the problem as canonical "physics objects" whose behavior is governed by physical laws. Such a model is always an approximation, since it will omit data which are unknown, omit interactions whose effects will be small, and use approximations of physical laws. The model which is constructed by the problem solver must account for all of the significant effects in the problem (otherwise the answer will be wrong) while omitting the insignificant effects (otherwise the problem will be unsolvable due to its complexity). We believe that the primary thing which is taught in physics classes is not the equations which express physical laws (these are often forgotten soon after the class is over), but the ability to "think physically", i.e., to view a new situation in terms of the canonical "physics objects" and to identify which factors will be important and which will be unimportant.

We believe that the process of analyzing a physics problem can be modeled using rules analogous to those which have been used in modeling expertise in tasks such as medical diagnosis [3]. That is, we view the process of problem solving in physics as being a diagnosis problem (in which the salient features of the

problem are diagnosed and an appropriate method of analysis is prescribed) rather than a process of logical deduction in the sense of mathematical theorem proving.

Problem analysis involves determining the general physical environment of the problem, filling in information missing from the problem statement, choosing analysis frameworks, and choosing physical models for objects and situations; each of these is discussed below.

It is necessary to determine the general physical environment because the environment helps determine which physical principles should be applied. For example, gravitational forces between planets are important, but electrostatic forces between planets are never considered; at the subatomic level, electrostatic forces are large but gravitational forces are insignificant. The general physical environment is never mentioned in problem statements, but must be inferred from the objects which are mentioned and the questions which are asked. For example, a problem involving an airplane will involve an earth-surface environment.

Physics problems as found in textbooks are stated very briefly, and do not explicitly include all the information which is needed for problem solving by either people or machines. It is necessary for the problem solver to fill in the missing information. For example, if the problem involves an airplane which drops a package, it is necessary to infer that before the package was dropped, it was held by the plane, and therefore had the same position and velocity as the plane. Our analysis of protocols of novice problem solving shows that novices often fail to fill in such information, and therefore have too little information to proceed with problem solving.

Effective problem solving requires that an appropriate coordinate framework for the problem be selected. Coordinate frameworks do not actually exist in the problem situation, but are "gedanken" constructs added by the problem solver. Use of an appropriate coordinate system can greatly simplify the equations used in the solution process. For example, appropriate choice of the zero point of the coordinate system will eliminate constants in the equations; proper alignment of coordinate axes will make features such as motions invariant along most coordinate axes. The rules for coordinate system choice and alignment are relatively simple; again, however, difficulty with these rules hinders novice problem solving.

Effective problem solving requires that objects and situations be modeled at an appropriate level of detail. While the laws of quantum mechanics presumably govern all physical systems, the equations which these laws involve are so complex that they can be solved only for the simplest systems at the atomic level. In order to solve everyday problems, approximations which give reasonably accurate answers must be used. For example, Newton's laws are approximations of relativistic mechanics which give good answers at low speeds. Choice of the physical models must be done on a qualitiative basis. For example, we know that a problem

involving an accelerating car can be solved using Newton's laws because cars do not normally move at relativistic speeds.

Matching the schemas for canonical "physics objects" and "physics situations" to the actual objects and situations found in problems is difficult. It is relatively easy to write a schema which will match any given problem. However, if it covers a whole problem, such a schema is unlikely to match many other problems, since other problems will undoubtedly have some different features. Use of large schemas therefore fails because a combinatoric number of schemas would be required. Use of small schemas is also difficult because the appropriate schema to use for part of the problem depends not only on local features but also on the global problem context. Problem analysis therefore requires the synthesis of global features from local information and the redistribution of the global information to the local level. In this sense problem analysis is like natural language parsing, in which sentence-level constraints apply to the phrases which make up the sentence. We have used the MRS language, which is based on first-order predicate calculus, to express problem analysis rules. This formalism allows flexible execution of rules when enough information becomes available for them to be used. We have successfully used these rules to analyze mechanics problems with what we believe is a robust and general formulation of the rules. A set of problem analysis rules is included as an appendix to this report.

5. Machine Learning of Problem Solving Skill Through Practice

In our original ISSAC [2] program for solving physics problems, a general formulation of the laws of rigid body statics, similar to that found in physics textbooks, was used. We were surprised to find that this program would generate a dozen equations for problems for which people would generate only one or two equations. It became clear, upon analyzing this phenomenon, that people were using special cases of the physical laws. These special-case methods reduce the number of equations by omitting equations which are not used, by making choices of coordinates which eliminate some equations, by combining several equations into one, and so forth.

We are investigating how a computer program can learn expertise in problem solving through practice. Our model of such learning is that the problem solver starts with a general formulation of physical laws, similar to those found in textbooks. The problem solver solves actual problems using the general laws; retrospective analysis of the solution process then allows specializations of the general laws, together with conditions for their use, to be generated. For the particular classes of problems they cover, the specialized forms of the laws will solve problems more efficiently than the general forms of the same laws. Generation of specialized laws is triggered upon recognition of particular features in the solution process. For example, generation of an equation which did not play a part in obtaining the solution to the problem indicates that the

equation could have been omitted. The occurrence of a free choice (for example, the choice of a point about which to sum moments in a rigid body statics problem) indicates that there might be a particularly appropriate choice in a given situation. Quantities which turn out to be small relative to the other quantities in the problem can be omitted. Symmetry conditions may permit special solution methods.

Work on this learning program is continuing as the Ph.D. thesis research of Agustin Araya; he is currently performing this research at the Pontificia Universidad Catolica de Chile, Santiago, Chile. Copies of his thesis abstract will be furnished at a later date.

6. Bibliography

[1] Genesereth, M.R., Greiner, R., and Smith, David E.
MRS Manual.
Technical Report HPP-80-24, Stanford Heuristic Programming Project, 1981.

[2] Novak, G. S.
Representations of Knowledge in a Program for Solving Physics Problems.
In *Proc. 5th Int'l Joint Conf. on Artificial Intelligence*, pages 286-291. M.I.T., Cambridge, MA, 1977.

[3] Shortliffe, E.H.
Computer Based Medical Consultations: MYCIN.
American Elsevier, 1976.

Publications

1. Novak, G., "Goals and Methodology of Research on Solving Physics Problems", Technical Report TR-158, Computer Science Department, University of Texas at Austin, Austin, TX, November, 1979.

2. Novak, G., "The Nature of Expert Problem Solving in Physics", Second Annual Conference of the Cognitive Science Society, Yale University, New Haven, CT, June, 1980. (Oral presentation; only abstracts were published.)

3. Novak, G. and Araya, A., "Research on Expert Problem Solving in Physics", *Proceedings of the First Annual National Conference on Artificial Intelligence* , Stanford, CA, August, 1980, pp. 178-180.

4. Novak, G., "GIRL and GLISP: An Efficient Representation Language", Technical Report TR-172, Computer Science Department, University of Texas at Austin, Austin, TX, March, 1981.

5. Novak, G. and Araya, A., "Physics Problem Solving Using Multiple Views", Technical Report TR-173, Computer Science Department, University of Texas at Austin, March, 1981.

6. Novak, G., "GLISP: An Efficient, English-Like Programming Language", *Proceedings of the Third Annual Conference of the Cognitive Science Society* , Berkeley, CA, August, 1981.

7. Novak, G., "Physics Problem Solving: ISAAC-II" (Program Demonstration), *Proceedings of the Seventh International Joint Conference on Artificial Intelligence* , Vancouver, B.C., Canada, August, 1981, p. 1063.

8. Novak, G., "GLISP User's Manual", Technical Report STAN-CS-82-895, Computer Science Department, Stanford University, Stanford, CA, January, 1982.

9. Novak, G., "Model Formulation for Physics Problem Solving", Working Paper, April, 1982.

10. Novak, G., "GLISP: A High-Level Language for A.I. Programming", *Proceedings of the Second National Conference on Artificial Intelligence* , Pittsburgh, PA, August, 1982.

11. Novak, G., "Data Abstraction in GLISP", submitted for publication, November 1982.

Data on Scientific Collaborators

Co-Investigator:

Jane Perlmutter, Ph.D., Assistant Professor of Psychology, University of Texas at Austin. Unfortunately, Dr. Perlmutter resigned from the University of Texas soon after the project began.

Research Assistants:

1. Agustin A. Araya, Ph.D. candidate, Computer Science Department, University of Texas at Austin.
2. Man-Lee Wan, Graduate Student, Computer Science Department, University of Texas at Austin.
3. Gildardo Zarza-Arronte, Graduate Student, Computer Science Department, University of Texas at Austin.
4. Sam Ellis, Graduate Student in Science Education, University of Texas at Austin.

Example of Knowledge Representation for Units of Measurement

```
(FILECREATED " 1-MAR-82 17:36:34" {DSK}MEASUREMENT.LSP;8 15862

     changes to:  MEASUREMENTCLASSES MEASUREMENTFNS MeasurementUnit.ConvertTo MEASUREMENTCOMS
MeasurementUnit.Dimensionality MeasurementUnit.Conversion DEFDERIVEDUNITS INITBASICUNITS
MeasurementUnit

     previous date: "25-FEB-82 15:14:18" {DSK}MEASUREMENT.LSP;3)


(PRETTYCOMPRINT MEASUREMENTCOMS)

(RPAQQ MEASUREMENTCOMS ((VARS MEASUREMENTCLASSES MEASUREMENTFNS)
                        (CLASSES * MEASUREMENTCLASSES)
                        (FNS * MEASUREMENTFNS)
                        (GLISPOBJECTS Dimensionality)))

(RPAQQ MEASUREMENTCLASSES (MeasurementUnit MassUnit LengthUnit TimeUnit ChargeUnit AngleUnit
                                           DimensionlessUnit ForceUnit AreaUnit VolumeUnit
                                           EnergyUnit PowerUnit))

(RPAQQ MEASUREMENTFNS (INITBASICUNITS DEFBASICUNITS DEFDERIVEDUNITS Dimensionality.+
                                      Dimensionality.- MeasurementUnit.ConvertTo
                                      MeasurementUnit.Dimensionality MeasurementUnit.Conversion
                                      MeasurementUnit.Prin1))
(DEFCLASSES MeasurementUnit MassUnit LengthUnit TimeUnit ChargeUnit AngleUnit DimensionlessUnit
            ForceUnit AreaUnit VolumeUnit EnergyUnit PowerUnit)
[DEFCLASS MeasurementUnit
   (MetaClass Class GETVALUE T)
   (Supers Object)
   (ClassVariables)
   (InstanceVariables (printNames NIL TYPE (LISTOF ATOM))
                      (expansion NIL TYPE (LIST (numerator (LISTOF MeasurementUnit))
                                                (denominator (LISTOF MeasurementUnit))))
                      (MKS-Conversion #((MeasurementUnit.Conversion self) FirstFetch NIL) TYPE
                                      REAL)
                      (dimensionality #((MeasurementUnit.Dimensionality self) FirstFetch NIL)
                                      TYPE Dimensionality))
   (Methods (Abbreviation ((CAR printNames)))
            (ConvertTo MeasurementUnit.ConvertTo RESULT Number)
            (Prin1 MeasurementUnit.Prin1))]

[DEFCLASS MassUnit
   (MetaClass Class)
   (Supers MeasurementUnit)
   (ClassVariables)
   (InstanceVariables (dimensionality 2852 TYPE Dimensionality))
   (Methods)]

[DEFCLASS LengthUnit
   (MetaClass Class)
   (Supers MeasurementUnit)
   (ClassVariables)
   (InstanceVariables (dimensionality 2404 TYPE Dimensionality))
   (Methods)]

[DEFCLASS TimeUnit
   (MetaClass Class)
   (Supers MeasurementUnit)
   (ClassVariables)
   (InstanceVariables (dimensionality 2348 TYPE Dimensionality))
   (Methods)]

[DEFCLASS ChargeUnit
   (MetaClass Class)
   (Supers MeasurementUnit)
   (ClassVariables)
   (InstanceVariables (dimensionality 2341 TYPE Dimensionality))
   (Methods)]

[DEFCLASS AngleUnit
   (MetaClass Class)
   (Supers MeasurementUnit)
   (ClassVariables)
   (InstanceVariables (dimensionality 2340 TYPE Dimensionality))
   (Methods)]
```

```
[DEFCLASS DimensionlessUnit
   (MetaClass Class)
   (Supers MeasurementUnit)
   (ClassVariables)
   (InstanceVariables (dimensionality 2340 TYPE Dimensionality))
   (Methods)]

[DEFCLASS ForceUnit
   (MetaClass Class)
   (Supers MeasurementUnit)
   (ClassVariables)
   (InstanceVariables)
   (Methods)]

[DEFCLASS AreaUnit
   (MetaClass Class)
   (Supers MeasurementUnit)
   (ClassVariables)
   (InstanceVariables)
   (Methods)]

[DEFCLASS VolumeUnit
   (MetaClass Class)
   (Supers MeasurementUnit)
   (ClassVariables)
   (InstanceVariables)
   (Methods)]

[DEFCLASS EnergyUnit
   (MetaClass Class)
   (Supers MeasurementUnit)
   (ClassVariables)
   (InstanceVariables)
   (Methods)]

[DEFCLASS PowerUnit
   (MetaClass Class)
   (Supers MeasurementUnit)
   (ClassVariables)
   (InstanceVariables)
   (Methods)]

(DEFINEQ

(INITBASICUNITS
  (LAMBDA NIL

        (* edited: " 1-MAR-82 11:15")

    (DEFBASICUNITS (QUOTE LengthUnit)
                   (QUOTE ((Foot .3048 (Ft Foot Feet))
                           (Meter 1.0 (M Meter Meters))
                           (Centimeter .01 (cm Centimeter Centimeters))
                           (Millimeter .001 (mm Millimeter Millimeters))
                           (Kilometer 1000.0 (Km Kilometer Kilometers))
                           (Micrometer 1.0E-6 (um Micrometer Micro-meter Micrometers
                                                  Micro-meters Micron Microns))
                           (Nanometer 1.0E-9 (nm Nanometer Nanometers))
                           (Angstrom 1.0E-10 (A Angstrom Angstroms))
                           (Inch .0254 (In Inch Inches))
                           (Mile 1609.344 (Mi Mile Miles))
                           (NauticalMile 1852.0 (NM NauticalMile NauticalMiles Nautical-Mile
                                                 Nautical-Miles))
                           (LightYear 9.46E15 (LY Light-Year Light-Years))
                           (Parsec 3.083E16 (Parsec Parsecs))
                           (Fathom 1.8054 (Fathom Fathoms))
                           (Yard .9144 (Yd Yard Yards))
                           (Rod 5.0292 (Rod Rods))
                           (Mil .0000254 (Mil Mils))
                           (Furlong 201.168 (Furlong Furlongs)))))
    (DEFBASICUNITS (QUOTE MassUnit)
                   (QUOTE ((Kilogram 1.0 (Kg Kilogram Kilograms))
                           (Gram .001 (Gm Gram Grams))
```

```
                        (MetricTon 2000.0 (Metric-Ton Metric-Tons Tonnes))
                        (Slug 14.5939 (Slug Slugs))
                        (AtomicMassUnit 1.66E-27 (AMU AMUs))
                        (Milligram .001 (mg Milligram Milligrams))
                        (Microgram 1.0E-6 (ug Microgram Micrograms)))))
    (DEFBASICUNITS (QUOTE TimeUnit)
                   (QUOTE ((Second 1.0 (S Sec Secs Second Seconds))
                        (Millisecond .001 (ms msec Millisecond Milliseconds))
                        (Microsecond 1.0E-6 (us usec Microsecond Microseconds))
                        (Nanosecond 1.0E-9 (ns nsec Nanosecond Nanoseconds))
                        (Picosecond 1.0E-12 (ps psec Picosecond Picoseconds))
                        (Minute 60.0 (min Minute Minutes))
                        (Hour 3600.0 (Hr Hour Hours))
                        (Day 86400.0 (Day Days))
                        (Week 604800.0 (Wk Week Weeks))
                        (Fortnight 1209600.0 (Fortnight Fortnights))
                        (Year 31557600.0 (Yr Year Years))
                        (Century 3.15576E9 (Century Centuries))
                        (Month 2629800.0 (Mon Month Months)))))
    (DEFBASICUNITS (QUOTE ChargeUnit)
                   (QUOTE ((Coulomb 1.0 (Coul Coulomb Coulombs))
                        (Abcoulomb 10.0 (Abcoul Abcoulomb Abcoulombs))
                        (Statcoulomb 3.336E-10 (Statcoul Statcoulomb Statcoulombs))
                        (AmpereHour 3600.0 (Amp-Hour Ampere-Hour Ampere-Hours))
                        (ElectronicCharge 1.602E-19 (Electronic-Charge)))))
    (DEFBASICUNITS (QUOTE AngleUnit)
                   (QUOTE ((Radian 1.0 (Radian Radians))
                        (Degree .0174533 (Deg Degree Degrees))
                        (ArcMinute .0002909 (ArcMin Arc-Minute Arc-Minutes))
                        (ArcSecond 4.848E-6 (ArcSec Arc-Second Arc-Seconds)))))
    (DEFBASICUNITS (QUOTE DimensionlessUnit)
                   (QUOTE ((Percent .01 (%% Percent))
                        (Exa 1.0E18 (Exa-))
                        (Peta 1.0E15 (Peta-))
                        (Tera 1.0E12 (Tera-))
                        (Giga 1.0E9 (Giga-))
                        (Mega 1000000.0 (Mega-))
                        (Kilo 1000.0 (Kilo-))
                        (Hecto 100.0 (Hecto-))
                        (Deka 10.0 (Deka- Deca-))
                        (Deci .1 (Deci-))
                        (Centi .01 (Centi-))
                        (Milli .001 (Milli-))
                        (Micro 1.0E-6 (Micro-))
                        (Nano 1.0E-9 (Nano-))
                        (Pico 1.0E-12 (Pico-))
                        (Femto 1.0E-15 (Femto-))
                        (Atto 1.0E-18 (Atto-)))))
    (DEFDERIVEDUNITS (QUOTE ForceUnit)
                     (QUOTE ((Pound ((Slug Foot)
                                     (Second Second))
                                    (Lb Lbs Pound Pounds))
                             (Newton ((Kilogram Meter)
                                      (Second Second))
                                     (Nt Newton Newtons))
                             (Dyne ((Gram Centimeter)
                                    (Second Second))
                                   (Dyne Dynes))
                             (Ounce ((Pound)
                                     (16))
                                    (Oz Ounce Ounces))
                             (Ton ((2000 Pound)
                                   NIL)
                                  (Ton Tons)))))
    (DEFDERIVEDUNITS (QUOTE AreaUnit)
                     (QUOTE ((MeterSquared ((Meter Meter)
                                            NIL)
                                           (M+2 Meter-Squared Square-Meter Square-Meters))
                             (CentimeterSquared ((Centimeter Centimeter)
                                                 NIL)
                                                (Cm+2 Centimeter-Squared Square-Centimeter
                                                      Square-Centimeters))
                             (FootSquared ((Foot Foot)
                                           NIL)
                                          (Ft+2 Foot-Squared Square-Foot Square-Feet))
```

```
                    (YardSquared ((Yard Yard)
                                  NIL)
                                 (Yard↑2 Yard-Squared Square-Yard Square-Yards))
                    (InchSquared ((Inch Inch)
                                  NIL)
                                 (In↑2 Inch-Squared Square-Inch Square-Inches))
                    (Hectare ((10000 MeterSquared)
                              NIL)
                             (Hectare))
                    (Acre ((43560 FootSquared)
                           NIL)
                          (Acre))
                    (MileSquared ((Mile Mile)
                                  NIL)
                                 (Mile↑2 Mile-Squared Square-Mile Square-Miles))
                    (KilometerSquared ((Kilometer Kilometer)
                                       NIL)
                                      (Km↑2 Kilometer-Squared Square-Kilometer
                                        Square-Kilometers))
                    (MillimeterSquared ((Millimeter Millimeter)
                                        NIL)
                                       (Mm↑2 Millimeter-Squared Square-Millimeter
                                         Square-Millimeters))
                    (MicrometerSquared ((Micrometer Micrometer)
                                        NIL)
                                       (uM↑2 Micrometer-Squared Micron-Squared
                                         Square-Micron Square-Microns)))))
(DEFDERIVEDUNITS (QUOTE VolumeUnit)
                 (QUOTE ((MeterCubed ((Meter Meter Meter)
                                      NIL)
                                     (M↑3 Meter-Cubed Cubic-Meter))
                         (CentimeterCubed ((Centimeter Centimeter Centimeter)
                                           NIL)
                                          (Cm↑3 Centimeter-Cubed Cubic-Centimeter
                                                Cubic-Centimeters Milliliter Milliliters
                                                mL))
                         (MillimeterCubed ((Millimeter Millimeter Millimeter)
                                           NIL)
                                          (Mm↑3 Millimeter-Cubed Cubic-Millimeter
                                                Cubic-Millimeters))
                         (KilometerCubed ((Kilometer Kilometer Kilometer)
                                          NIL)
                                         (Km↑3 Kilometer-Cubed Cubic-Kilometer
                                               Cubic-Kilometers))
                         (InchCubed ((Inch Inch Inch)
                                     NIL)
                                    (In↑3 Inch-Cubed Cubic-Inch Cubic-Inches))
                         (FootCubed ((Foot Foot Foot)
                                     NIL)
                                    (Ft↑3 Foot-Cubed Cubic-Foot Cubic-Feet))
                         (YardCubed ((Yard Yard Yard)
                                     NIL)
                                    (Yd↑3 Yard-Cubed Cubic-Yard Cubic-Yards))
                         (Mile-Cubed ((Mile Mile Mile)
                                      NIL)
                                     (Mile↑3 Mile-Cubed Cubic-Mile Cubic-Miles))
                         (AcreFoot ((Acre Foot)
                                    NIL)
                                   (Acre-Foot Acre-Feet))
                         (Liter ((.001 MeterCubed)
                                 NIL)
                                (L Liter Liters))
                         (Deciliter ((Liter)
                                     (10))
                                    (dl Deciliter Deciliters))
                         (Gallon ((3.785 Liter)
                                  NIL)
                                 (gal Gallon Gallons))
                         (Quart ((.946 Liter)
                                 NIL)
                                (qt Quart Quarts))
                         (Fifth ((Gallon)
                                 (5))
                                (Fifth Fifths))
                         (Pint ((.473 Liter)
```

```
                                    NIL)
                                   (pt Pint Pints))
                            (Cup ((Pint)
                                  (2))
                                 (Cup Cups))
                            (FluidOunce ((.029573 Liter)
                                         NIL)
                                        (FlOz Fluid-Ounce Fluid-Ounces))
                            (Tablespoon ((FluidOunce)
                                         (2))
                                        (tbsp Tablespoon Tablespoons))
                            (Teaspoon ((Tablespoon)
                                       (3))
                                      (tsp Teaspoon Teaspoons)))))
(DEFDERIVEDUNITS (QUOTE PowerUnit)
                 (QUOTE ((Watt ((Kilogram Meter Meter)
                                (Second Second Second))
                               (W Watt Watts))
                         (Milliwatt ((Milli Watt)
                                     NIL)
                                    (mW Milli-Watt Milli-Watts))
                         (Microwatt ((Micro Watt)
                                     NIL)
                                    (uW Micro-Watt Micro-Watts))
                         (Kilowatt ((Kilo Watt)
                                    NIL)
                                   (KW Kilowatt Kilowatts))
                         (Megawatt ((Mega Watt)
                                    NIL)
                                   (MW Megawatt Megawatts Mega-Watt Mega-Watts))
                         (Gigawatt ((Giga Watt)
                                    NIL)
                                   (GW Gigawatt Gigawatts Giga-Watt Giga-Watts))
                         (Horsepower ((745.701 Watt)
                                      NIL)
                                     (HP Horsepower)))))
(DEFDERIVEDUNITS (QUOTE EnergyUnit)
                 (QUOTE ((Joule ((Kilogram Meter Meter)
                                 (Second Second))
                                (J Joule Joules))
                         (FootPound ((Foot Pound)
                                     NIL)
                                    (FtLb Foot-Pound Foot-Pounds))
                         (KilowattHour ((Kilo Watt Hour)
                                        NIL)
                                       (KWH Kilowatt-Hour Kilowatt-Hours))
                         (WattHour ((Watt Hour)
                                    NIL)
                                   (WattHour Watt-Hour Watt-Hours))
                         (HorsepowerHour ((Horsepower Hour)
                                          NIL)
                                         (HP-Hour))
                         (ElectronVolt ((1.6021E-19 Joule)
                                        NIL)
                                       (EV Electron-Volt Electron-Volts))
                         (MEV ((1.6021E-13 Joule)
                               NIL)
                              (MEV Mega-Electron-Volts))
                         (GEV ((1.6021E-10 Joule)
                               NIL)
                              (GEV Giga-Electron-Volts))
                         (Calorie ((4.186 Joule)
                                   NIL)
                                  (cal calorie calories))
                         (KiloCalorie ((4186.0 Joule)
                                       NIL)
                                      (KCal Kilo-Calorie Kilo-Calories))
                         (BritishThermalUnit ((1055 Joule)
                                              NIL)
                                             (BTU BTUs))
                         (Erg ((1.0E-7 Joule)
                               NIL)
                              (Erg Ergs)))))))

(DEFBASICUNITS
```

```
  (GLAMBDA (CLASSNAME ARGUMENTS:(LISTOF (LIST (INSTNAME ATOM)
                                              (FACTOR REAL)
                                              (PRINTNAMES (LISTOF ATOM)))))

          (* edited: "25-FEB-82 15:10")


          (* Define Basic Measurement Units)

           (PROG (CLASSREC INST:MeasurementUnit)
                 (OR (CLASSREC ←(GetObjectRec CLASSNAME))
                     (ERROR CLASSNAME " is not a defined Class."))
                 (For each ARGUMENT Do (INST ←(← CLASSREC New))
                      (← INST SetName INSTNAME)
                      (INST:MKS-Conversion ← FACTOR)
                      (INST:printNames ← PRINTNAMES)))))

(DEFDERIVEDUNITS
  (GLAMBDA (CLASSNAME ARGUMENTS:(LISTOF (LIST (INSTNAME ATOM)
                                              (EXPANSION (LIST (NUMERATOR (LISTOF ATOM))
                                                               (DENOMINATOR (LISTOF ATOM))))
                                              (PRINTNAMES (LISTOF ATOM)))))

          (* edited: "26-FEB-82 11:29")


          (* Define derived Measurement Units)

           (PROG (CLASSREC INST:MeasurementUnit)
                 (OR (CLASSREC ←(GetObjectRec CLASSNAME))
                     (ERROR CLASSNAME " is not a defined Class."))
                 (For each ARGUMENT Do (INST ←(← CLASSREC New))
                      (← INST SetName INSTNAME)
                      (INST:expansion ←(LIST (For U in (the NUMERATOR)
                                                collect
                                                (If U is numeric then U else (GetObjectRec
                                                                                U)))
                                             (For U in (the DENOMINATOR)
                                                collect
                                                (If U is numeric then U else (GetObjectRec
                                                                                U)))))
                      (INST:printNames ← PRINTNAMES)))))

(Dimensionality.+
  (GLAMBDA (X,Y:Dimensionality)

          (* edited: "24-FEB-82 15:45")


           (X:self + Y:self - 2340)))

(Dimensionality.-
  (GLAMBDA (X,Y:Dimensionality)

          (* edited: "24-FEB-82 15:45")


           (X:self - Y:self + 2340)))

(MeasurementUnit.ConvertTo
  (GLAMBDA (GIVENUNIT,DESIREDUNIT:MeasurementUnit)

          (* edited: "24-FEB-82 15:35")


          (* "GSN: " "11-Dec-81 12:21")


           (PROG (F DIM)
                 (F ← GIVENUNIT:MKS-Conversion / DESIREDUNIT:MKS-Conversion)
```

```
                (DIM ← GIVENUNIT:dimensionality - DESIREDUNIT:dimensionality)
                (RETURN (If DIM is Zero then F elseif DIM is WeightToMass then F/9.81 elseif
                            DIM is MassToWeight then F*9.81 elseif DIM is MassToEnergy then
                            F*9.0E16 elseif DIM is EnergyToMass then F/9.0E16 else NIL)))))

(MeasurementUnit.Dimensionality
  (GLAMBDA (self:MeasurementUnit)

          (* edited: "26-FEB-82 09:44")


          (* Compute the Dimensionality of a MeasurementUnit from its expansion).

           (PROG (DIM:Dimensionality)
                 (DIM ← 2340)
                 (For U in (the numerator of the expansion of self)
                      (If U is not numeric then DIM←+U:dimensionality))
                 (For U in (the denominator of the expansion of self)
                      (If U is not numeric then DIM←-U:dimensionality))
                 (RETURN DIM))))

(MeasurementUnit.Conversion
  (GLAMBDA (self:MeasurementUnit)

          (* edited: "26-FEB-82 09:44")


          (* Compute the MKS-Conversion for a MeasurementUnit from its expansion)

           (PROG ((F 1.0))
                 (For U in (the numerator of the expansion of self)
                      (F ← F *(If U is numeric then U else U:MKS-Conversion)))
                 (For U in (the denominator of the expansion of self)
                      (F ← F /(If U is numeric then U else U:MKS-Conversion)))
                 (RETURN F))))

(MeasurementUnit.Prin1
  (GLAMBDA (self:MeasurementUnit)

          (* edited: "25-FEB-82 10:17")


           (PRIN1 (CAR printNames))))
)

[GLDEFSTRQ

(Dimensionality

   (self INTEGER)

   ADJ    ((Zero (self = 2340))
           (WeightToMass (self = 2388))
           (MassToWeight (self = 2292))
           (MassToEnergy (self = 2228))
           (EnergyToMass (self = 2452)))

   MSG    ((+ Dimensionality.+ RESULT Dimensionality OPEN T)
           (- Dimensionality.- RESULT Dimensionality OPEN T))  )
]

(DECLARE: DONTCOPY
  (FILEMAP (NIL (3457 15497 (INITBASICUNITS 3467 . 12023) (DEFBASICUNITS 12025 . 12588) (
DEFDERIVEDUNITS 12590 . 13454) (Dimensionality.+ 13456 . 13587) (Dimensionality.- 13589 . 13720)
 (MeasurementUnit.ConvertTo 13722 . 14310) (MeasurementUnit.Dimensionality 14312 . 14845) (
MeasurementUnit.Conversion 14847 . 15355) (MeasurementUnit.Prin1 15357 . 15495)))))
STOP
```

```
:    CARPROB.MRS.13    14 July 82

:(*  CARPROB.MRS :  Simon & Simon car problem.
:    Gordon S. Novak Jr.     29 March 82     )


(DEFINEASSERTIONS '(

(CURRENT-PROBLEM PROBLEM1)
(SITUATION SIT0)
(EVENT EVENT1)
(SITUATION SIT1)
(EVENT EVENT2)
(END-SITUATION SIT0 EVENT1)
(BEGIN-SITUATION EVENT1 SIT1)
(END-SITUATION SIT1 EVENT2)

(CAR CAR1)
(CAR CAR2)
(AT-REST CAR1 SIT0)
(PASS CAR2 CAR1 EVENT1)
(PASS CAR1 CAR2 EVENT2)

(VALUE CAR2 VELOCITY CAR2VEL SIT0)

(MEASUREMENT CAR2VEL)
(NUMVAL CAR2VEL 28)
(UNITS CAR2VEL M//SEC)
(MEASURE CAR2VEL VELOCITY)
(TYPE-MEASURE CAR2VEL SCALAR)

(VALUE CAR1 ACCELERATION CAR1ACC SIT1)

(MEASUREMENT CAR1ACC)
(NUMVAL CAR1ACC 4)
(UNITS CAR1ACC M//SEC2)
(MEASURE CAR1ACC ACCELERATION)
(TYPE-MEASURE CAR1ACC SCALAR)

(QUESTION (TIME-OF EVENT2))


))
```

```
;   AIRPROB.MRS.6        14 July 82

:(* AIRPROB.MRS          Initially Created 12 April 82.)


;(* Airplane Problem, from Halliday & Resnick p. 57)

:(* A plane is flying at a constant horizontal velocity of 500 km/h at
;   an elevation of 5.0 km toward a point directly above its target.  At
;   what angle of sight phi should a survival package be released to
;   strike the target?)


(DEFINEASSERTIONS '(

(CURRENT-PROBLEM PROBLEM2)
(AIRPLANE AIRPLANE1)
(SITUATION SIT0)
(VALUE AIRPLANE1 VELOCITY VEL1 SIT0 T)
(FLYING AIRPLANE1 SIT0)
(ELEVATION AIRPLANE1 ALTITUDE1 SIT0)
(EVENT EVENT1)
(INITIAL-EVENT PROBLEM2 EVENT1)
(END-SITUATION SIT0 EVENT1)
(PACKAGE PACKAGE1)
(RELEASE AIRPLANE1 PACKAGE1 EVENT1)
(EVENT EVENT2)
(TARGET TARGET1)
(HIT PACKAGE1 TARGET1 EVENT2)


:(* Things that should eventually be inferred.)

(SITUATION SIT1)
(BEGIN-SITUATION EVENT1 SIT1)
(EVENT EVENT2)
(END-SITUATION SIT1 EVENT2)
(VALUE AIRPLANE1 POSITION POS0 EVENT1)

(MEASUREMENT VEL1)
(NUMVAL VEL1 500)
(UNITS VEL1 KM//H)
(HORIZONTAL VEL1)

(MEASUREMENT ALTITUDE1)
(NUMVAL ALTITUDE1 5)
(UNITS ALTITUDE1 KM)


))
```

```
;   PHYSRULES.MRS.41        15 August 82

;(* "PHYSRULES.MRS :  Physics rules in MRS format.
;   Gordon S. Novak Jr.     29 March 1982        ")



(DEFINERULES '(

(RULE100 (PROBLEM-REPRESENTATION)
(* "Make the current problem a situation.")
(IF (CURRENT-PROBLEM $PROB)
    (SITUATION $PROB))
FO)

(RULE101 (PROBLEM-REPRESENTATION)
(* "Make the current problem a sub-situation of reality.")
(IF (CURRENT-PROBLEM $PROB)
    (SUB-SITUATION $PROB REALITY))
FO)

(RULE102 (ANALYSIS)
(* "Make each new situation which is asserted a sub-situation of
   the problem, if it is not a sub-situation of something else.")
(IF (AND (SITUATION $SIT)
         (THNOT (SUB-SITUATION $SIT $SIT2))
         (THNOT (EQUAL $SIT REALITY))
         (CURRENT-PROBLEM $PROB))
    (SUB-SITUATION $SIT $PROB))
FO)

(RULE103 (PROBLEM-REPRESENTATION)
(* "If a parameter is constant in a situation, and the value
   of the parameter is known for any time, it has that value
   for all times.")
(IF (AND (GROUND $TIME2)
         (CONSTANT $OBJ $PARAM $SIT)
         (VALUE $OBJ $PARAM $VAL $SIT))
    (VALUE $OBJ $PARAM $VAL $SIT $TIME2))
BO)

(RULE104 (PROBLEM-REPRESENTATION)
(* "If a parameter has a value in a surrounding situation,
   and the parameter is normally constant, inherit the value
   from the surrounding situation.")
(IF (AND (GROUND $OBJ)
         (GROUND $PARAM)
         (GROUND $SUBSIT)
         (USUALLY-CONSTANT $OBJ $PARAM)
         (VALUE $OBJ $PARAM $VAL $SIT)
         (SUB-SITUATION $SUBSIT $SIT) )
    (VALUE $OBJ $PARAM $VAL $SUBSIT))
)

(RULE106 (PROBLEM-REPRESENTATION)
(* "Sub-Situation is transitive.")
(IF (AND (SUB-SITUATION $SIT1 $SIT2)
         (SUB-SITUATION $SIT2 $SIT3))
    (SUB-SITUATION $SIT1 $SIT3))
FO)

(RULE108 (MECHANICS)
(* "Position is continuous between situations.")
(IF (AND (PHYSICAL-OBJECT $OBJ)
         (SITUATION-SUCCESSOR $SIT1 $EVENT $SIT2))
    (CONTINUOUS $OBJ POSITION $EVENT))
)
```

```
(RULE109  (MECHANICS)
(* "Velocity is continuous between situations unless
   there is a collision.")
(IF (AND (PHYSICAL-OBJECT $OBJ)
         (SITUATION-SUCCESSOR $SIT1 $EVENT $SIT2)
         (THNOT (COLLIDES $OBJ $OBJB $EVENT)) )
    (CONTINUOUS $OBJ VELOCITY $EVENT))
)

(RULE110 (MECHANICS)
(* "If a parameter of an object is continuous between situations,
   and the value of the parameter is known at the time of the
   transition between situations, the value is the same at that
   time in the other situation.")
(IF (AND (CONTINUOUS $OBJ $PARAM $EVENT)
         (SITUATION-SUCCESSOR $SIT1 $EVENT $SIT2)
         (TIME-OF $EVENT $TIME)
         (VALUE $OBJ $PARAM $VAL $SIT1 $TIME))
    (VALUE $OBJ $PARAM $VAL $SIT2 $TIME))
)

; ??? Better to let EQUAL take care of this.
(RULE110B (MECHANICS)
(* "If a parameter of an object is continuous between situations,
   and the value of the parameter is known at the time of the
   transition between situations, the value is the same at that
   time in the other situation.")
(IF (AND (CONTINUOUS $OBJ $PARAM $EVENT)
         (SITUATION-SUCCESSOR $SIT1 $EVENT $SIT2)
         (TIME-OF $EVENT $TIME)
         (VALUE $OBJ $PARAM $VAL $SIT2 $TIME))
    (VALUE $OBJ $PARAM $VAL $SIT1 $TIME))
)

(RULE111 (CAUSALITY)
(* "Situations are connected by events.")
(IF (AND (END-SITUATION $SIT1 $EVENT)
         (BEGIN-SITUATION $EVENT $SIT2)
         (EVENT $EVENT)
         (SITUATION $SIT1)
         (SITUATION $SIT2))
    (SITUATION-SUCCESSOR $SIT1 $EVENT $SIT2))
FO)

(RULE112 (CAUSALITY)
(* "Constant velocity implies zero acceleration.")
(IF (CONSTANT $OBJ VELOCITY $SIT)
    (VALUE $OBJ ACCELERATION ZERO $SIT))
FO)

(RULE113 (CAUSALITY)
(* "An object at rest has zero velocity.")
(IF (AT-REST $OBJ $SIT)
    (VALUE $OBJ VELOCITY ZERO $SIT))
FO)

(RULE114 (MECHANICS)
(* "When one object passes another, their positions are the same.")
(IF (PASS $OBJ1 $OBJ2 $EVENT)
    (SAME $OBJ1 POSITION $OBJ2 POSITION $EVENT))
)

(RULE115 (PROBLEM-REPRESENTATION)
(* "SAME is commutative.")
(IF (SAME $OBJ1 $PARAM1 $OBJ2 $PARAM2 $EVENT)
    (SAME $OBJ2 $PARAM2 $OBJ1 $PARAM1 $EVENT))
)

(RULE116  (ALGEBRA)
```

```
(* "If a value is constant, it is known.")
(IF (AND (VALUE $OBJ $PARAM $VAL $SIT $TIME)
         (CONSTANT-VALUE $VAL))
    (KNOWN $OBJ $PARAM $VAL $SIT $TIME))
FO)

(RULE117 (ALGEBRA)
(* "If a value is known and it is equal to another value, set
   the other value.")
(IF (AND (KNOWN $OBJ $PARAM $VAL $SIT $TIME)
         (EQUAL $OBJ $PARAM $SIT $TIME
                $OBJ2 $PARAM2 $SIT2 $TIME2))
    (VALUE $OBJ2 $PARAM2 $VAL $SIT2 $TIME2))
FO)

(RULE118 (PROBLEM-REPRESENTATION)
(* "A parameter whose value is numeric has constant value.")
(IF (AND (MEASUREMENT $VAL)
         (NUMVAL $VAL $V)
         (NUMERIC $V))
    (CONSTANT-VALUE $VAL))
)

(RULE119 (ALGEBRA)
(* "A measurement with numeric value 0 is zero.")
(IF (AND (MEASUREMENT $VAL)
         (NUMVAL $VAL 0))
    (ZERO $VAL))
)

(RULE120 (ANALYSIS)
(* "An object with nonzero velocity moves.")
(IF (AND (PHYSICAL-OBJECT $OBJ)
         (GROUND (VALUE $OBJ VELOCITY $VAL $SIT $TIME))
         (THNOT (ZERO $VAL)))
    (MOVES $OBJ $SIT))
)

(RULE120B (ANALYSIS)
(* "An object with nonzero velocity moves.")
(IF (AND (PHYSICAL-OBJECT $OBJ)
         (GROUND (VALUE $OBJ VELOCITY $VAL $SIT))
         (THNOT (ZERO $VAL)))
    (MOVES $OBJ $SIT))
)

(RULE121 (ANALYSIS)
(* "An object with nonzero acceleration accelerates")
(IF (AND (PHYSICAL-OBJECT $OBJ)
         (GROUND (VALUE $OBJ ACCELERATION $VAL $SIT $TIME))
         (THNOT (ZERO $VAL)))
    (ACCELERATES $OBJ $SIT))
)

(RULE121B (ANALYSIS)
(* "An object with nonzero acceleration accelerates")
(IF (AND (PHYSICAL-OBJECT $OBJ)
         (GROUND (VALUE $OBJ ACCELERATION $VAL $SIT))
         (THNOT (ZERO $VAL)))
    (ACCELERATES $OBJ $SIT))
)

(RULE121C (ANALYSIS)
(* "An object which accelerates moves.")
(IF (ACCELERATES $OBJ $SIT)
    (MOVES $OBJ $SIT))
)

(RULE122 (ANALYSIS)
```

```
(* "An aircraft which is flying moves.")
(IF (AND (AIRCRAFT $X)
         (FLYING $X $SIT)
         (SITUATION $SIT))
    (MOVES $X $SIT))
FO)

(RULE123 (ANALYSIS)
(* "Something which is falling accelerates.")
(IF (AND (ORDINARY-OBJECT $X)
         (FALLING $X $SIT)
         (SITUATION $SIT))
    (ACCELERATES $X $SIT))
FO)

(RULE124 (ANALYSIS)
(* "If an object passes another object, the passer moves in the
   situation which precedes the passing event.")
(IF (AND (PHYSICAL-OBJECT $OBJ)
         (PASS $OBJ $OBJ2 $EVENT)
         (END-SITUATION $SIT $EVENT))
    (MOVES $OBJ $SIT))
FO)

(RULE125 (ANALYSIS)
(* "If a situation is bounded by two events, the second event
   is after the first.")
(IF (AND (BEGIN-SITUATION $EVENT1 $SIT)
         (END-SITUATION $SIT $EVENT2))
    (AFTER $EVENT2 $EVENT1))
FO)

(RULE126 (ANALYSIS)
(* "After is transitive.")
(IF (AND (AFTER $EVENT1 $EVENT2)
         (AFTER $EVENT2 $EVENT3))
    (AFTER $EVENT1 $EVENT3))
FO)



:(* "Rules for environment and coordinate systems.")

(RULE180 (ANALYSIS)
(* "If there is an event which begins a situation, and it is not
   after another event, it is an initial event.")
(IF (AND (BEGIN-SITUATION $EVENT $SIT)
         (THNOT (AFTER $EVENT $EARLIEREVENT))
         (CURRENT-PROBLEM $PROB))
    (INITIAL-EVENT $PROB $EVENT))
)

(RULE181 (ANALYSIS)
(* "If the value of a parameter has been asserted for a situation
   without mention of time, the value is constant in that situation.")
(IF (AND (VALUE $OBJ $PARAM $VAL $SIT)
         (CONSTANT-VALUE $VAL))
    (CONSTANT $OBJ $PARAM $SIT))
FO)

(RULE201 (ANALYSIS)
(* "If the problem contains a surface vehicle, assume
   the environment is earth-surface.")
(IF (AND (SURFACE-VEHICLE $X)
         (CURRENT-PROBLEM $PROB)
         (THNOT (ENVIRONMENT $PROB $ENV)) )
```

```
    (ENVIRONMENT $PROB EARTH-SURFACE))
OS FO)

(RULE202 (ANALYSIS)
(* "If the problem contains an aircraft, assume
   the environment is earth-surface.")
(IF (AND (AIRCRAFT $X)
         (FLYING $X $SIT)
         (CURRENT-PROBLEM $PROB)
         (SUB-SITUATION $SIT $PROB))
    (ENVIRONMENT $PROB EARTH-SURFACE))
OS FO)

(RULE203 (ANALYSIS)
(* "If the problem involves a surface vehicle which moves,
   a horizontal coordinate will be needed.")
(IF (AND (SURFACE-VEHICLE $X)
         (MOVES $X $SIT)
         (CURRENT-PROBLEM $PROB)
         (SUB-SITUATION $SIT $PROB))
    (CS-DIR-NEEDED $PROB HORIZONTAL))
FO)

(RULE204 (ANALYSIS)
(* "If the problem involves something falling, a vertical
   coordinate will be needed.")
(IF (AND (PHYSICAL-OBJECT $X)
         (FALLING $X $SIT)
         (CURRENT-PROBLEM $PROB)
         (SUB-SITUATION $SIT $PROB))
    (CS-DIR-NEEDED $PROB VERTICAL))
FO)

(RULE205 (ANALYSIS)
(* "If both horizontal and vertical coordinates are needed,
   select a vertical plane as coordiante system.")
(IF (AND (CS-DIR-NEEDED $PROB HORIZONTAL)
         (CS-DIR-NEEDED $PROB VERTICAL))
    (SELECT-CS $PROB VERTICAL-PLANE-CS))
)

(RULE206 (ANALYSIS)
(* "If only a horizontal coordinate is needed, select a horizontal
   linear coordinate system.")
(IF (AND (CURRENT-PROBLEM $PROB)
         (UNIQUE (CS-DIR-NEEDED $PROB $DIR))
         (EQUAL $DIR HORIZONTAL))
    (SELECT-CS $PROB HORIZONTAL-LINEAR-CS))
)

(RULE207 (ANALYSIS)
(* "If only a vertical coordinate is needed, select a vertical
   linear coordinate system.")
(IF (AND (CURRENT-PROBLEM $PROB)
         (UNIQUE (CS-DIR-NEEDED $PROB $DIR))
         (EQUAL $DIR VERTICAL))
    (SELECT-CS $PROB VERTICAL-LINEAR-CS))
)

(RULE207B (ANALYSIS)
(IF (HORIZONTAL-LINEAR-CS $CS)
    (LINEAR-CS $CS))
FO)

(RULE207C (ANALYSIS)
(IF (VERTICAL-LINEAR-CS $CS)
    (LINEAR-CS $CS))
FO)
```

```
(RULE208 (ANALYSIS)
(* "If a coordinate system is selected, make one.")
(IF (AND (CURRENT-PROBLEM $PROB)
         (SELECT-CS $PROB $CSTYPE)
         (GENOBJECT $CSTYPE $CS))
    (COORDINATE-SYSTEM $PROB $CS))
OS FO)

(RULE209 (ANALYSIS)
(* "If anything moves, consider it a dynamics problem.")
(IF (AND (CURRENT-PROBLEM $PROB)
         (SUB-SITUATION $SIT $PROB)
         (MOVES $OBJ $SIT))
    (DYNAMIC $PROB))
OS)

(RULE210 (ANALYSIS)
(* "If the problem is not dynamic, it is static.")
(IF (AND (CURRENT-PROBLEM $PROB)
         (THNOT (DYNAMIC $PROB)))
    (STATIC $PROB))
OS)

(RULE211 (ANALYSIS)
(* "If a problem is dynamic, it involves time.")
(IF (AND (CURRENT-PROBLEM $PROB)
         (DYNAMIC $PROB))
    (INVOLVES-TIME $PROB))
OS)

(RULE212 (ANALYSIS)
(* "If a problem involves time, set up a time scale for it.")
(IF (AND (CURRENT-PROBLEM $PROB)
         (INVOLVES-TIME $PROB)
         (THNOT (TIMESCALE $PROB $TS1))
         (GENOBJECT TIMESCALE $TS))
    (TIMESCALE $PROB $TS))
OS FO)

(RULE213 (ANALYSIS)
(* "If a specific time is mentioned, note that it is a time reference
point for this problem.")
(IF (AND (GROUND (TIME-OF $EVENT $TIME))
         (EVENT $EVENT)
         (CONSTANT-VALUE $TIME)
         (CURRENT-PROBLEM $PROB))
    (TIME-REFERENCE $PROB $TIME))
)

(RULE214 (ANALYSIS)
(* "If an event doesn't have a time, make one.")
(IF (AND (CURRENT-PROBLEM $PROB)
         (INVOLVES-TIME $PROB)
         (EVENT $EVENT)
         (THNOT (TIME-OF $EVENT $TIME))
         (GENOBJECT TIME $TIME1))
    (TIME-OF $EVENT $TIME1))
)

(RULE215 (ANALYSIS)
(* "If there is no specific time mentioned, choose the initial event
   as the zero point of the time scale.")
(IF (AND (CURRENT-PROBLEM $PROB)
         (INVOLVES-TIME $PROB)
         (THNOT (TIME-REFERENCE $PROB $TIME))
         (UNIQUE (INITIAL-EVENT $PROB $EVENT1))
         (TIMESCALE $PROB $TS))
    (ZERO-POINT $TS $EVENT1))
OS)
```

```
(RULE216 (ANALYSIS)
(* "If the zero-point of a time scale is chosen to be a particular
   event, set the time of that event to be zero.")
(IF (AND (CURRENT-PROBLEM $PROB)
         (TIMESCALE $PROB $TS)
         (ZERO-PJINT $TS $EVENT)
         (TIME-OF $EVENT $TIME))
    (VALUE $TIME ZERO))
OS)

(RULE217 (ANALYSIS)
(* "An object with constant velocity is in uniform motion.")
(IF (AND (CONSTANT $OBJ VELOCITY $SIT)
         (MOVES $OBJ $SIT)
         (SITUATION $SIT))
    (UNIFORM-MOTION $OBJ $SIT))
FO)

(RULE218 (ANALYSIS)
(* "An object with constant acceleration is in uniform
   acceleration.")
(IF (AND (CONSTANT $OBJ ACCELERATION $SIT)
         (VALUE $OBJ ACCELERATION $VAL $SIT)
         (THNOT (ZERO $VAL))
         (SITUATION $SIT))
    (UNIFORM-ACCELERATION $OBJ $SIT))
FO)

(RULE219 (ANALYSIS)
(* "An object which is undergoing uniform acceleration and was at
   rest in the preceding situation is in uniform acceleration from
   rest.")
(IF (AND (UNIFORM-ACCELERATION $OBJ $SIT)
         (SITUATION-SUCCESSOR $SIT1 $EVENT $SIT)
         (AT-REST $OBJ $SIT1))
    (UNIFORM-ACCELERATION-FROM-REST $OBJ $SIT))
FO)

(RULE220 (ANALYSIS)
(* "Try to make the zero-point of the coordinate system the same as
   the position of some significant event which occurs at the initial
   time.")
(IF (AND (COORDINATE-SYSTEM $PROB $CS)
         (CURRENT-PROBLEM $PROB)
         (DYNAMIC $PROB)
         (THNOT (ZERO-POINT $CS $PT))
         (TIMESCALE $PROB $TS)
         (ZERO-POINT $TS $EVENT)
         (EVENT $EVENT))
    (ZERO-POSITION-EVENT $CS $EVENT))
FO)



:(* "Rules for causality.")

(RULE301 (CAUSALITY)
(* "If an aircraft is flying and it releases something, and the
   thing which is released is not an aircraft, then assume that
   the released object is dropped.")
(IF (AND (GROUND (RELEASE $AIRCRAFT $OBJ $EVENT))
         (AIRCRAFT $AIRCRAFT)
         (ORDINARY-OBJECT $OBJ)
         (THNOT (AIRCRAFT $OBJ))
         (END-SITUATION $SIT $EVENT)
         (FLYING $AIRCRAFT $SIT))
    (DROP $AIRCRAFT $OBJ $EVENT))
```

```
)

(RULE302 (CAUSALITY)
(* "If an event involves dropping an object, in the following
   situation the object will be falling.")
(IF (AND (GROUND (DROP $OBJ1 $OBJ2 $EVENT))
         (BEGIN-SITUATION $EVENT $SIT))
    (FALLING $OBJ2 $SIT))
)

(RULE303 (ANALYSIS)
(* "If an object is falling in a situation and the event at the
   end of the situation involves the object hitting something,
   and the elevation of the object which is hit is not known,
   assume the target object is at ground level.")
(IF (AND (GROUND (FALLING $OBJ $SIT))
         (END-SITUATION $SIT $EVENT)
         (HIT $OBJ $TARGET $EVENT)
         (THNOT (ELEVATION $TARGET $ELEV $SIT)) )
    (VERTICAL-LOCATION $TARGET EARTH-SURFACE))
)

(RULE304 (CAUSALITY)
(* "If an event involves dropping an object, in the situation
   preceding the dropping event, the dropper was holding
   the dropped object.")
(IF (AND (GROUND (DROP $OBJ1 $OBJ2 $EVENT))
         (END-SITUATION $SIT $EVENT))
    (HOLD $OBJ1 $OBJ2 $SIT))
)

(RULE304B (CAUSALITY)
(* "If an object is held, it and the holder are rigidly attached.")
(IF (HOLD $OBJ1 $OBJ2 $SIT)
    (RIGIDLY-ATTACHED $OBJ1 $OBJ2 $SIT))
FO)

(RULE305 (CAUSALITY)
(* "If an object is rigidly attached during a situation, its position
   at the end of the situation is the same as that of the object to which
   it is attached.")
(IF (AND (GROUND (RIGIDLY-ATTACHED $OBJ1 $OBJ2 $SIT))
         (END-SITUATION $SIT $EVENT)
         (TIME-OF $EVENT $TIME))
    (SAME $OBJ1 POSITION $OBJ2 POSITION $EVENT))
)

(RULE306  (CAUSALITY)
(* "If an object is rigidly attached during a situation, its velocity
   at the end of the situation is the same as that of the object to which
   it is attached.")
(IF (AND (GROUND (RIGIDLY-ATTACHED $OBJ1 $OBJ2 $SIT))
         (END-SITUATION $SIT $EVENT)
         (TIME-OF $EVENT $TIME))
    (SAME $OBJ1 VELOCITY $OBJ2 VELOCITY $EVENT))
)

(RULE307 (MECHANICS)
(* "If an object is falling, its acceleration is downward with
   the value of gravitational acceleration.")
(IF (AND (FALLING $OBJ $SIT)
         (SITUATION $SIT)
         (PHYSICAL-OBJECT $OBJ)
         (CURRENT-PROBLEM $PROB)
         (ENVIRONMENT $PROB $ENV)
         (GRAVITY $ENV $G))
    (VALUE $OBJ ACCELERATION $G $SIT T))
)
```

```
(RULE308 (MECHANICS)
(IF (AND (FALLING $OBJ $SIT)
         (SITUATION $SIT)
         (PHYSICAL-OBJECT $OBJ)
         (CURRENT-PROBLEM $PROB)
         (ENVIRONMENT $PROB $ENV)
         (GRAVITY $ENV $G))
    (DIRECTION $OBJ ACCELERATION DOWNWARD $SIT))
)

(RULE309 (CAUSALITY)
(* "Altitude of an aircraft means distance above the ground.")
(IF (AND (AIRCRAFT $PLANE)
         (VALUE $PLANE ALTITUDE $VAL $SIT $TIME))
    (ABOVE $PLANE GROUND $VAL $SIT $TIME))
FO)

(RULE310 (CAUSALITY)
(* "If an aircraft is in level flight, its vertical velocity component
   is zero.")
(IF (AND (AIRCRAFT $PLANE)
         (FLIGHT $PLANE LEVEL $SIT)
         (VALUE $PLANE VELOCITY $VAL $SIT $TIME))
    (VERTICAL-COMPONENT $VAL ZERO))
)

(RULE311 (CAUSALITY)
(* "If two objects are at the same position at the beginning and end
   of a situation, and they move during the situation, they must be
   moving in a similar direction.")
(IF (AND (SAME $OBJ1 POSITION $OBJ2 POSITION $EVENT1)
         (BEGIN-SITUATION $EVENT1 $SIT)
         (END-SITUATION $SIT $EVENT2)
         (SAME $OBJ1 POSITION $OBJ2 POSITION $EVENT2)
         (MOVES $OBJ1 $SIT)
         (MOVES $OBJ2 $SIT))
    (SIMILAR-MOTION-DIRECTION $OBJ1 $OBJ2 $SIT))
)

(RULE312 (ANALYSIS)
(* "If an object was moving with constant velocity in a situation,
   and its velocity is continuous into another situation, and we
   don't know that it is accelerating in the new situation,
   it will move with the same constant velocity in the new situation.")
(IF (AND (CONSTANT $OBJ VELOCITY $SIT)
         (SITUATION-SUCCESSOR $SIT $EVENT $SIT2)
         (CONTINUOUS $OBJ VELOCITY $EVENT)
         (THNOT (ACCELERATES $OBJ $SIT2))
         (VALUE $OBJ VELOCITY $VEL $SIT))
    (VALUE $OBJ VELOCITY $VEL $SIT2))
)

(RULE313 (ANALYSIS)
(* "If an event involves one object passing another, make a position
   for that event.")
(IF (AND (PASS $OBJ1 $OBJ2 $EVENT)
         (EVENT $EVENT))
    (MAKE-EVENT-POSITION $OBJ1 $OBJ2 $EVENT))
FO)

(RULE314 (ANALYSIS)
(* "If a position is to be made for objects in an event, and
   neither object has a position, make one.")
(IF (AND (MAKE-EVENT-POSITION $OBJ1 $OBJ2 $EVENT)
         (THNOT (VALUE $OBJ1 POSITION $POS1 $EVENT))
         (THNOT (VALUE $OBJ2 POSITION $POS2 $EVENT))
         (GENOBJECT POSITION $P))
    (ASSERT-POSITION-CREATION $OBJ1 $P $EVENT))
FO)
```

```
(RULE314B (ANALYSIS)
(* "If position is created, set object position value.")
(IF (ASSERT-POSITION-CREATION $OBJ1 $P $EVENT)
    (VALUE $OBJ1 POSITION $P $EVENT))
FO)
(RULE314C (ANALYSIS)
(IF (ASSERT-POSITION-CREATION $OBJ1 $P $EVENT)
    (MEASUREMENT $P))
FO)
(RULE314D (ANALYSIS)
(IF (ASSERT-POSITION-CREATION $OBJ1 $P $EVENT)
    (MEASURE $P POSITION))
FO)

(RULE315 (ANALYSIS)
(* "If a value is known for a parameter, and it is known that the
   parameter is the same as another parameter, set the parameter
   value.")
(IF (AND (SAME $OBJ1 $PARAM1 $OBJ2 $PARAM2 $EVENT)
         (VALUE $OBJ1 $PARAM1 $V $EVENT))
    (VALUE $OBJ2 $PARAM2 $V $EVENT))
FO)

(RULE316 (ANALYSIS)
(* "If a zero-position event has been selected for a coordinate system,
   and no zero-position has been established for the coordinate system,
   and a position of some object in the event is known, select that
   position as the zero-point of the coordinate system.")
(IF (AND (ZERO-POSITION-EVENT $CS $EVENT)
         (THNOT (ZERO-POSITION $CS $VAL))
         (COORDINATE-SYSTEM $PROB $CS)
         (CURRENT-PROBLEM $PROB)
         (VALUE $OBJ POSITION $V $EVENT))
    (ZERO-POSITION $CS $V))
FO)

(RULE317 (ANALYSIS)
(* "If a zero-position has been selected for a linear coordinate system,
   and no numeric value has been assigned to that position, assign it
   the value 0.")
(IF (AND (ZERO-POSITION $CS $V)
         (COORDINATE-SYSTEM $PROB $CS)
         (CURRENT-PROBLEM $PROB)
         (THNOT (NUMVAL $V $N)) )
    (NUMVAL $V 0))
FO)

(RULE318 (ANALYSIS)
(* "If a zero-position is selected for a coordinate system, mark the
   position as relative to that coordinate system.")
(IF (AND (ZERO-POSITION $CS $V)
         (COORDINATE-SYSTEM $PROB $CS)
         (CURRENT-PROBLEM $PROB))
    (COORDINATE-REFERENCE $V $CS))
FO)

(RULE319 (ANALYSIS)
(* "If a zero-position is selected for a linear coordinate system,
   mark the type of the variable as scalar.")
(IF (AND (ZERO-POSITION $CS $V)
         (COORDINATE-SYSTEM $PROB $CS)
         (CURRENT-PROBLEM $PROB)
         (LINEAR-CS $CS))
    (TYPE-MEASURE $V SCALAR))
FO)
```

```
:(* "Rules for problem analysis.")

(RULE401 (ANALYSIS)
(* "Make the question a goal.")
(IF (QUESTION $P)
    (GOAL $P))
)

(RULE402 (ANALYSIS)
(* "If a goal involves the time of an event, and two objects
   pass each other at that event, try to express the position
   of each object as a function of time, and solve simultaneously.")
(IF (AND (GOAL (TIME-OF $EVENT))
         (EVENT $EVENT)
         (PASS $OBJ1 $OBJ2 $EVENT)
         (END-SITUATION $SIT $EVENT)
         (TIME-OF $EVENT $TIME))
     (GOAL $OBJ1 POSITION $SIT $TIME))
)
(RULE402B (ANALYSIS)
(IF (AND (GOAL (TIME-OF $EVENT))
         (EVENT $EVENT)
         (PASS $OBJ1 $OBJ2 $EVENT)
         (END-SITUATION $SIT $EVENT)
         (TIME-OF $EVENT $TIME))
     (GOAL $OBJ2 POSITION $SIT $TIME))
)

(RULE403 (ANALYSIS)
(* "If a goal is to find when an object hits something,
   and the thing hit is a horizontal surface, try to find the
   time when the vertical position of the object equals that
   of the surface.")
(IF (AND (GOAL (TIME-OF $EVENT))
         (EVENT $EVENT)
         (HIT $OBJ1 $OBJ2 $EVENT)
         (SURFACE $OBJ2)
         (STATIONARY $OBJ2)
         (HORIZONTAL $OBJ2)
         (END-SITUATION $SIT $EVENT)
         (TIME-OF $EVENT $TIME))
    (EQUAL-GOAL ($OBJ1 VERTICAL-POSITION $SIT $TIME)
                ($OBJ2 VERTICAL-POSITION)) )
)

(RULE404 (WRITE-EQUATIONS)
(* "If an object is in uniform motion along a coordinate direction,
   and its position is a goal, express it using (d - d0) = v*(t-t0).")
(IF (AND (GOAL $OBJ POSITION $SIT $TIME)
         (UNIFORM-MOTION $OBJ $SIT)
         (BEGIN-SITUATION $IEVENT $SIT)
         (TIME-OF $IEVENT $ITIME)
         (VALUE $OBJ POSITION $IPOS $IEVENT)
         (TIME-OF $EVENT $TIME)
         (VALUE $OBJ POSITION $POS $EVENT)
         (VALUE $OBJ VELOCITY $VEL $SIT)
         (BC (TIME-DIFFERENCE $TIME $ITIME $DELTAT))
         (BC (POSITION-DIFFERENCE $POS $IPOS $DELTAP)))
    (MAKE-EQUATION (EQUAL $DELTAP (TIMES $VEL $DELTAT))) )
)

(RULE405 (WRITE-EQUATIONS)
(* "If initial time is zero, time difference is just final time.")
(IF (AND (GROUND (TIME-OF $EVENT $ITIME))
         (VALUE $ITIME ZERO))
    (TIME-DIFFERENCE $TIME $ITIME $TIME))
80)

(RULE406 (WRITE-EQUATIONS)
```

```
(* "If initial time is not zero, make a difference operator.")
(IF (AND (GROUND (TIME-OF $EVENT $ITIME))
         (THNOT (VALUE $ITIME ZERO)))
    (TIME-DIFFERENCE $TIME $ITIME (DIFFERENCE $TIME $ITIME)) )
BO)

(RULE407 (WRITE-EQUATIONS)
(* "If initial position is zero, position difference is just final
   position.")
(IF (AND (MEASUREMENT $IPOS)
         (GROUND $IPOS)
         (ZERO $IPOS))
    (POSITION-DIFFERENCE $POS $IPOS $POS))
BO)

(RULE408 (WRITE-EQUATIONS)
(* "If initial position is not zero, make a difference operator.")
(IF (AND (MEASUREMENT $IPOS)
         (GROUND $IPOS)
         (THNOT (ZERO $IPOS)))
    (POSITION-DIFFERENCE $POS $IPOS (DIFFERENCE $POS $IPOS)) )
BO)

(RULE409 (ANALYSIS)
(* "If two objects have similar motion direction during a situation,
   and the coordinate system is linear, they have the same motion
   direction.")
(IF (AND (SIMILAR-MOTION-DIRECTION $OBJ1 $OBJ2 $SIT)
         (CURRENT-PROBLEM $PROB)
         (COORDINATE-SYSTEM $PROB $CS)
         (LINEAR-CS $CS))
    (SAME-MOTION-DIRECTION $OBJ1 $OBJ2 $SIT))
)



;(* "Things that ordinarily remain constant.")

(RULE501 (CAUSALITY)
(IF (ORDINARY-OBJECT $OBJ) (USUALLY-CONSTANT $OBJ MASS))
)
(RULE502 (CAUSALITY)
(IF (CELESTIAL-OBJECT $OBJ) (USUALLY-CONSTANT $OBJ MASS))
)


;(* "Taxonomic relations.")
(RULE601 (TAXONOMY) (IF (CAR $X) (SURFACE-VEHICLE $X)) FO)
(RULE602 (TAXONOMY) (IF (SURFACE-VEHICLE $X) (TERRESTRIAL-VEHICLE $X)) FO)
(RULE603 (TAXONOMY) (IF (TERRESTRIAL-VEHICLE $X) (VEHICLE $X)) FO)
(RULE604 (TAXONOMY) (IF (AIRPLANE $X) (AIRCRAFT $X)) FO)
(RULE605 (TAXONOMY) (IF (AIRCRAFT $X) (VEHICLE $X)) FO)
(RULE606 (TAXONOMY) (IF (VEHICLE $X) (ORDINARY-OBJECT $X)) FO)
(RULE607 (TAXONOMY) (IF (PACKAGE $X) (ORDINARY-OBJECT $X)) FO)
(RULE608 (TAXONOMY) (IF (ORDINARY-OBJECT $X) (PHYSICAL-OBJECT $X)) FO)
(RULE609 (TAXONOMY) (IF (PLANET $X) (CELESTIAL-OBJECT $X)) FO)
(RULE610 (TAXONOMY) (IF (STAR $X) (CELESTIAL-OBJECT $X)) FO)
(RULE611 (TAXONOMY) (IF (CELESTIAL-OBJECT $X) (PHYSICAL-OBJECT $X)) FO)


))
: End of DEFINERULES

(DEFINEASSERTIONS '(
;(* "Numeric constants.")

(SITUATION REALITY)
```

```
(MEASUREMENT ZERO)
(NUMVAL ZERO 0)
(UNITS ZERO T)
(TYPE-MEASURE ZERO SCALAR)
(ZERO ZERO)

(GRAVITY EARTH-SURFACE EARTH-GRAVITY)
(MEASUREMENT EARTH-GRAVITY)
(NUMVAL EARTH-GRAVITY 9.8)
(UNITS EARTH-GRAVITY M/SEC2)
(TYPE-MEASURE EARTH-GRAVITY SCALAR)


))
```